Neural Networks in Text Processing

Natural Language Processing

Fernando André Bezerra Moura Fernandes - bez0070

06-01-2020

Contents

1 Abstract

Text Processing allows the automation of creating and manipulating electronic text with numerous modern applications such as grammar and spell checking, text completion, plagiarism detection or even chat bots. Thus, the theory behind text processing techniques requires a lot of **algorithmic** and **data structure** knowledge.

These techniques allow the development of artificial intelligence applications and normally have to deal with large quantities of strings when analyzing textual documents for their different uses.

One very specific use case of textual analysis is **Natural Language Processing** which is the one of the two main topics of this document.

Natural Language Processing is an artificial intelligence field concerned with interactions between computers and a language (i.e English) and the processing of textual data in that language so that the machine can make conclusions about that document.

In order to understand a document written in some language a computer program must first pre-process that document so that later it can be modeled in a computable way. Common ways of doing this include:

- **Tokenizing the document** by breaking down the document's text into different words and representing a document in memory by its words. One example is to divide the document by the space character and getting the words that way.
- **Stemming the words** which means to get the base word and removing the derivational affixes.
- **Lemmatization** by removing the inflectional ending of the words and getting the root word ensuring it belongs to that language.
- **N-Grams**. N-grams refer to the process of combining the nearby words together for representation purposes where N represents the number of words to be combined together.

In order to model a document by its content, first a meaning must be given to the words it contains.

One way of doing this is calculating a **TF-IDF** value for each word on a document, which can be interpreted as its **weight**. Another way would be calculating **One-Hot Encodings** or **Word Embeddings** which are other ways of representing words in a document. After these transformations the document can be represented in a certain format such as a **Bag of Words**.

All these concepts and techniques and much more are used by **Deep Neural Networks** in the field of Natural Language Processing. **Neural Networks** are a set of algorithms which are modeled to resemble the way that the human brain works. Deep Neural Networks are a subfield of Neural Networks which use multiple layers to progressively extract higher level features of the input they are given. **DNN** have revolutionized the field of Natural Language Processing and are the second main topic of this document.

2 Introduction

Neural networks in the area of computer science are mostly referred to as **Artificial Neural Networks** and are based on the neural structure of the brain, being able to learn to perform tasks such as classification and clustering.
An **ANN** consists of a series of nodes, which are interpreted as artificial neurons which are part of a directed and weighted graph and can be set on an input layer, output layer, or on the middle hidden processing layer or layers.

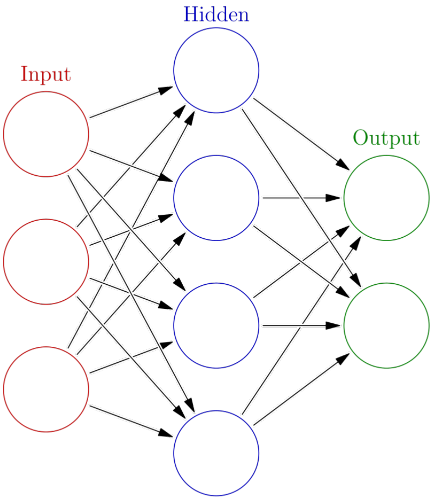

Figure 1: A representation of an ANN

The input layer receives input, parses it and sends information to the hidden layer(s) which in turn sends it to the output layer. Every neuron(node) has **synapses**(input), which they combine with their internal state using an **activation function** and produce an output. The input to a neuron is the output of other neurons or simply another neuron, which is computed by a **propagation function**. An ANN also possesses **hyperparameters** which are parameters who control the network, are set before the learning process and change accordingly via learning. Examples of these parameters include the **learning rate** or the number of hidden layers and can depend on other hyperparameters.
The ANN consumes a dataset, considering it as a group of sample observations and uses one part of the dataset(train) to be trained and to learn, varying its hyperparameters and another part (test) to check its accuracy. **Learning** involves adjusting the weights of the network to decrease an error rate. Usually this **error rate** is divided among the connections and **backpropagation** is used to adjust the connection weights. Backpropagation is a method who calculates the **gradient** of the **cost function**(which measures the performance of the ANN regarding its test set) and then the weight updates are often performed using **stochastic gradient descent** or other methods. Depending on their internal organization, or the way they change their weights there are many different

types of ANN, and throughout the rest of this research document, their uses on Natural Language Processing will be described.

3 Deep Learning on Natural Language Processing

Depending on the NLP task at hand we can use different artificial neural network models and achieve different results. First we must discuss to how features in a document should be represented.

Generally speaking, in NLP the sparse-input encodes features such as words or part-of-speech tags. The initial task is to parse this input into a neural-network base model and start representing each feature as a **dense vector**. Each feature is **embedded** into a dimensional space, and represented as a vector in that space, and is treated as a model parameter which needs to be trained together with other components of the network [4]. One benefit of using dense and low-dimensional vectors is computational efficiency since the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors. The other benefit of the dense representations is in generalization power because as some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities.

In cases where we have relatively few distinct features in the category, and we believe there are no correlations between the different features, we may use the one-hot representation vectors. The one-hot representation vectors result in a sparse feature vector where each dimension is a feature and feature combinations receive their own dimension, and where values inside the vector are binary resulting in high dimensionality. Dense feature representations, represent each feature as a vector resulting in low dimensionality and the feature-to-vector mapping is made on an **embedding matrix**. There is also another problem with features related with the number in which they occur, as in some cases their number is not known in advance. Thus, an unbounded number of features needs to be represented using a fixed size vector. One way of doing this is using a **continuous bag of words**. The traditional bag of words is a way to represent the data in a tabular format with columns representing the total vocabulary of the documents, rows representing a single observation and each cell being filled up by the observation frequency. The CBOW is very similar to the traditional bag-of-words representation in which we discard order information, and works by either summing or averaging the embedding vectors of the corresponding features.

$$CBOW(f_1, ..., f_k) = \frac{1}{k}\sum_{i=1}^{k} v(f_i) \tag{1}$$

Now, we will describe a series of different neural network architectures and the way they are using on NLP tasks.

3.1 Feed Forward Neural Networks

A **feed forward neural network** is an acyclic type of neural network architecture, which means the connections between the nodes do not form any cycles in the whole network.

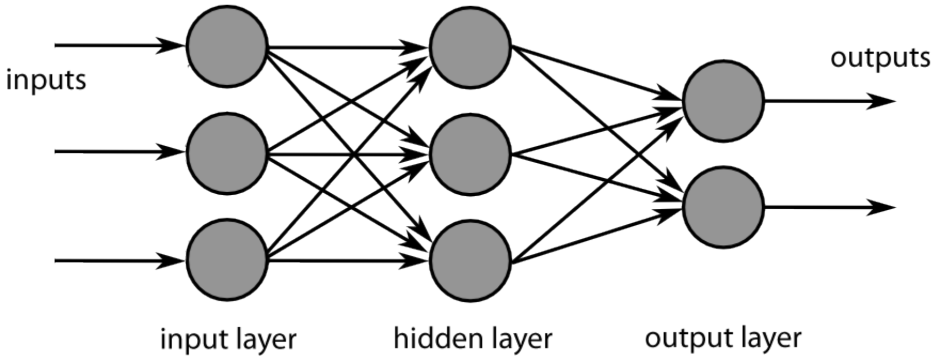

Figure 2: A representation of a Feed Forward Neural Network

As we can see from figure 2 the information is carried always in one direction from the input to the output. Using mathematical notations, each layer in the network implements a vector-matrix multiplication $\mathbf{h} = \mathbf{xW}$ where W represents the weights matrix and x represents the layer vector(nodes in that layer). The values of h are then transformed using a linear function g before being passed to the next layer. We define the whole computation as follows: $g(xW_1)\prod_{i=2}^{n} W_i$, where n is the number of hidden layers and W_i represents the weight matrix of the ith layer.

3.1.1 Multi Layer Perceptron

A **Multi Layer Perceptron** is a class of feed forward neural networks and its architecture is composed of multiple layers of **perceptrons**. Perceptrons are the simplest neural network and are considered binary classifiers whose algorithm is based on a linear prediction function.

$$Perceptron(x) = xW + b \tag{2}$$

The term b is the bias. The multi layer perceptron is able to go beyond linear functions by adding more perceptron layers. Example with 2 hidden layers:

$$MLP_2(x) = (g_2(g_1(xW_1 + b_1)W_2 + b_2))W_3 \tag{3}$$

The vector resulting from each linear transform is referred to as a layer. The outer-most linear transform results in the output layer and the other linear transforms result in hidden layers. Each hidden layer is followed by a non-linear activation.

In an NLP application, x is normally composed of embedding vectors, wether they are one-hot representations or dense representations. One common task in NLP is classification, and one frequently used method used in text classification is the multilayer perceptron.

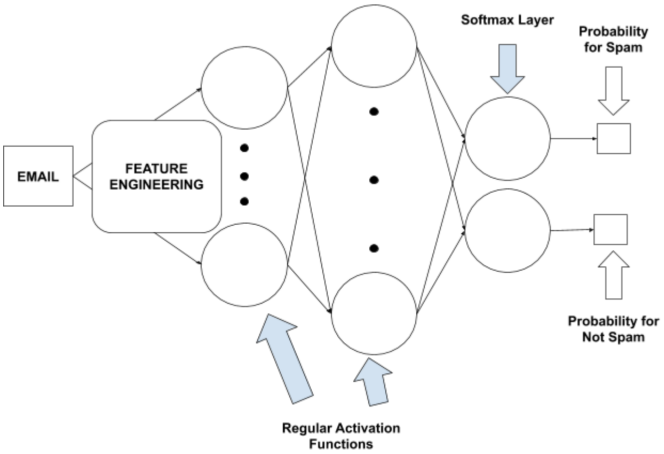

Figure 3: Email spam classification with a MLP

First, a set of features are derived from an e-mail text, which are feeded to an input layer and then results from that layer are propagated into the hidden middle layers. There are also cases where the first layer of the network is responsible for feature engineering. In this case the output layer will yield a size 2 vector with probabilities for the *Spam* and *Not spam* classes. The **Softmax Layer** contains a series of neurons whose activation function is the softmax function which converts a set of class scores into equivalent class probabilities.

$$\sigma(x)_j = \frac{e^{xj}}{\sum_{i=1}^{k} e^{xi}} \tag{4}$$

3.1.2 Convolutional Neural Networks

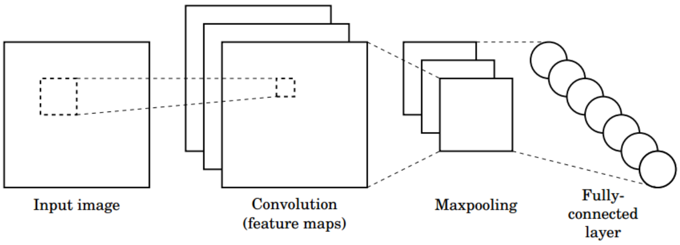

Figure 4: Representation of a CNN

Convolutional Neural Networks use **convolution** in place of matrix multiplication on at least on of their layers.
The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a **rectifier** layer, and is subsequently followed by additional convolutions such as **pooling** layers, **fully connected layers** and **normalization** layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product. Convolution is a mathematical operation on two functions that produces a third function expressing how the shape of one is modified by the other. On the previous section, when describing the MLP model, the e-mail spam classifier was mentioned. On this section we will make reference to a papper, *"Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts"* [3], on which the authors propose a convolutional neural network architecture for a NLP task which extracts sentiment analysis from textual data. Their proposed deep neural network is called **CharSCNN**, *Character to Sentence Convolutional Neural Network*, and uses 2 convolutional layers to extract relevant features from words and sentences of any size. They focus their work on small textual data, such as movie reviews or *Twitter* messages. Given a sentence, the network computes a score for each sentiment label. In order to score a sentence, the network takes as input the sequence of words in the sentence, and passes it through a sequence of layers where features with increasing levels of complexity are extracted. The first layer of the network transforms words into real-valued feature vectors (embeddings) that capture morphological, syntactic and semantic information about the word. The sentences are represented using a set of N words $w_1, w_2, w_3, ..., w_n$ and every word w_n is converted into a vector $u_n = [r^{wrd}, r^{wch}]$. The first element of the vector is the **word-level embedding** and the second one is the **character level-embedding**. Word-level embeddings are encoded by column vectors in an embedding matrix $W^{wrd} \in \mathbb{R}^{d^{wrd} \times |V^{wrd}|}$ The word is transformed into its word-level embedding by:

$$r^{wrd} = W^{wrd} v^{w} \tag{5}$$

For character-level embedding, considering a word to be represented by M characters $c_1, c_2, c_3, ..., c_m$ the character embedding is:

$$r^{chr} = W^{chr} v^c \tag{6}$$

$W^{chr} \in \mathbb{R}^{d^{chr} \times |V^{chr}|}$ is the character embedding matrix. They define the next step in CharSCNN to be extracting a sentence-level representation r_x^{sent}. The second convolutional layer produces local features around each word in the sentence and then combines them using a *max* operation to create fixed-sized feature vectors for the word sequence(sentence). The network is trained by minimizing a negative likelihood over a training set.
This is just an example of how CNN's can be used for NLP tasks.

3.2 Recurrent Neural Networks

Recurrent Neural Networks is a variant of **recursive neural networks** characterized by being a directed cyclic graph. They use their internal state, stored in memory, to process sequences of inputs, being extremely useful for cases where input order matters, like in most NLP applications. In *"Recurrent Neural Networks for Language Understanding"* [5], the authors use a RNN architecture for LU tasks such as labeling words with semantic meaning. They use the classic **Elman** architecture for this purpose. The Elman architecture is a three-layer network, where the middle (hidden) layer is connected to the context units(u in the illustration) fixed with a weight of one. At each time step, the input is fed forward and a learning rule is applied. The fixed back-connections save a copy of the previous values of the hidden units in the context units (since they propagate over the connections before the learning rule is applied). Thus the network can maintain a sort of state, allowing it to perform such tasks as sequence-prediction. The authors represent U, W and V as the weight matrices of the 3 layers, $w(t)$ as the input word at time t on the form of **1-of-N encoding**, $y(t)$ as the probability distribution function over semantic labels produced by the output layer and $s(t)$ is used on the middle layer

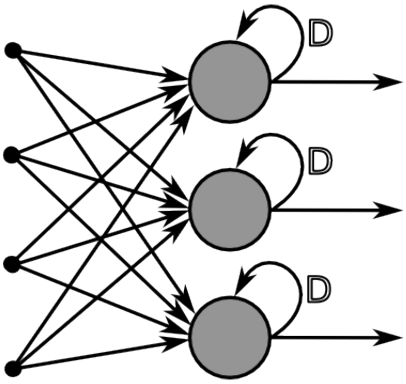

Figure 5: Representation of the RNN's recurrent layer

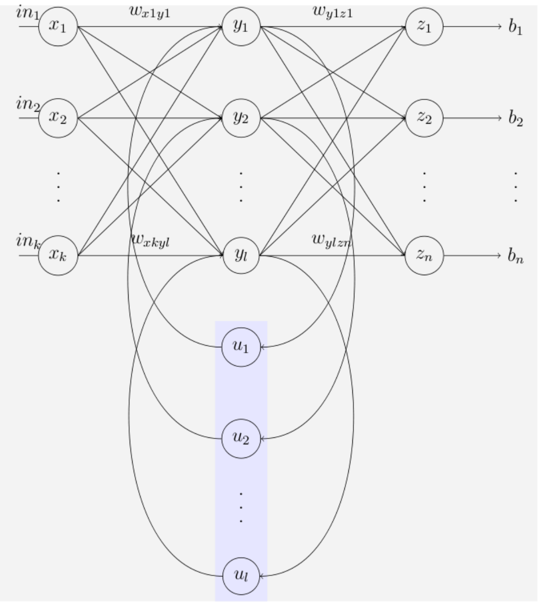

Figure 6: Representation of the Elman architecture

to maintain a representation of sentence history.

$$
\begin{aligned}
s(t) &= f(Uw(t) + Ws(t-1)) \\
y(t) &= g(Vs(t)) \\
f(z) &= \frac{1}{1+e^{-z}} \\
g(z_m) &= \frac{e^{z_m}}{\sum_k e^{z_k}}
\end{aligned}
$$

The model is trained using standard backpropagation to maximize the data conditional likelihood. Then, the authors change the way words are encoded in the first layer to a bag-of-words,$x(t)$, in which there is a non-zero value for not just the current word, but the next n - 1 words as well and change the side information of the network to dense vectors $f(t)$ instead of one-hot representations via another hidden layer. The new hidden layer is connected to the middle one and the output one and the connection weights are defined as F and G respectively.

$$
\begin{aligned}
s(t) &= f(Ux(t) + Ws(t-1) + Ff(t)) \\
y(t) &= g(Vs(t) + Gf(t)) \\
x(t) &= w(t), w(t+1)
\end{aligned}
$$

Actually $x(t)$ can also be $w(t)$ or the defined bag of words vector.

3.2.1 Long Short-Term Memory Neural Networks

LSTM's are a type of RNN architecture with four layers, the **cell**, the **input gate**, the **output gate**, and the **forget gate**. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

- The input gate controls information input at each time step
- The output gate controls how much information is outputted to the next cell
- The forget gate controls how much information is lost at each time step

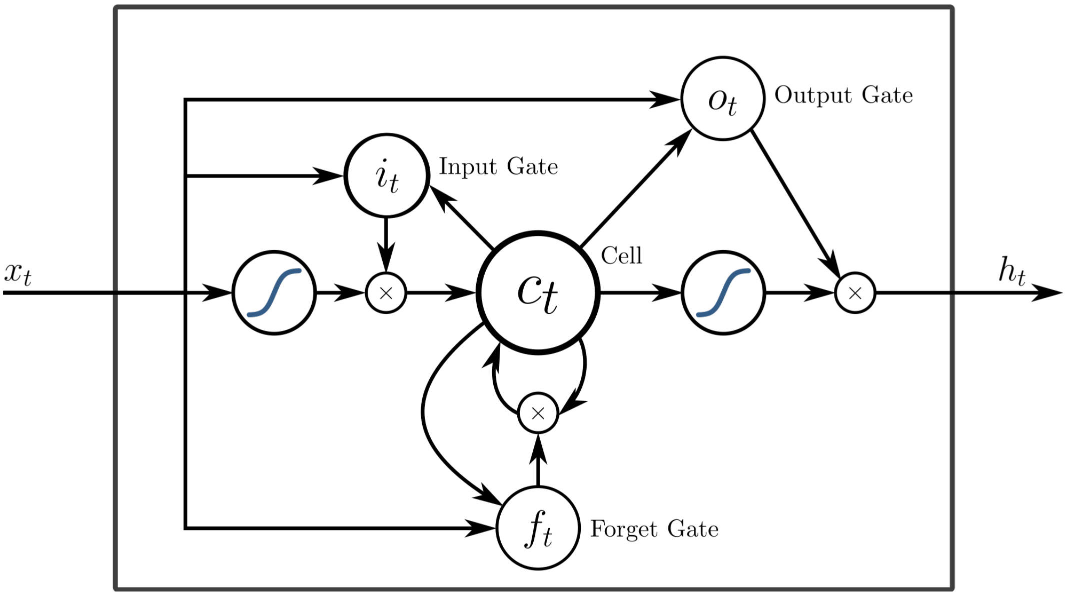

Figure 7: A representation of a LSTM

In *"Long Short-Term Memory Neural Networks for Chinese Word Segmentation"* [2], there is a practical application of the use of LSTM's for an NLP task. It is a natural choice to apply LSTM neural networks to word segmentation tasks since LSTM's can learn from data with long range temporal dependencies due to the considerable time lag between the inputs and their corresponding outputs. **Word segmentation** is fundamental for the chinese language processing, and the authors propose a LSTM to tackle this problem. Their proposal architecture contains a first layer responsible for character embeddings: Considering the character dictionary as C of size $|C|$, each character $c \in C$ is represented as a real sized vector $v_c \in \mathbb{R}^d$, where d is the dimensionality of the vector space. These character embeddings are stacked into the embedding matrix, $M \in \mathbb{R}^{d \times |C|}$, and they are retrieved via a lookup table layer.

Their LSTM's memory cell, c, encodes memory at every time step of whatever inputs have been observed up to the step and its behavior is controlled by the three LSTM's architectural gates mentioned above, i, f, o. Definition of the gates and cell updates:

$$\begin{aligned}
\sigma(x) &= (1 + e^{-x})^-1 \\
\Phi(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\
i(t) &= \sigma(W_{ix}x(t) + W_{ih}h(t-1) + W_{ic}c(t-1)) \\
c(t) &= f(t)c(t-1) + i(t)\Phi(W_{cx}x(t) + W_{ch}h(t-1)) \\
o(t) &= \sigma(W_{ox}x(t) + W_{oh}h(t-1) + W_{oc}c(t)) \\
h(t) &= o(t)\Phi(c(t))
\end{aligned}$$

They also propose 4 more slightly different LSTM architectures, *LSTM-1*, *LSTM-2*, *LSTM-3*, *LSTM-4*, that differ on the number of layers and are not covered here on this report. Their LSTM is trained using the **Max-Margin criterion**. For a given sentence x_i, $Y(x_i)$ is the set of all possible tag sequences and the correct tag sequence is y_i. For a given training instance (x_i, y_i) the predicted tag sequence $\hat{y_i} \in Y(x_i)$ is:

$$\hat{y_i} = argmaxs(x_i, y, \theta) \tag{7}$$

Here, s is related to the **sentence level score**: The structured margin loss would be:

$$\Delta(y_i, \hat{y}) = \sum_t^n \rho \mathbf{1} \{y_i(t) \neq \hat{y}(t)\} \tag{8}$$

Then giving a training set D, the regularized objective function is:

$$\begin{aligned}
J(\theta) &= \frac{1}{D} \sum_{(x_i,y_i)\in D} +l_i(\theta) + \frac{\lambda}{2}||\theta|| \\
l_i(\theta) &= max(0, s(x_i, \hat{y_i}, \theta) + \Delta(y_i, \hat{y_i}) - s(x_i, y_i, \theta))
\end{aligned}$$

θ is the parameter set. In order to, minimize $J(\theta)$, they use the **subgradient** method:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{r=1}^t g_{r,i}^2 g_{t,i}}} \tag{9}$$

α is the initial learning rate, and g_x is the subgradient at time step x.

4 Conclusions

Contrasting with the old use of machine learning methods, which used linear models such as support vector machines or logistic regression, on a very sparse and highly dimensional input, the NLP field has been modified by the use of neural network technology as we've seen. The most common architectures for natural language processing tasks fall in the area of recurrent neural networks. This is explained due to the fact that they can store internal state in memory and therefore are useful in cases where the sequence of the input is very important such as in tasks related with text generation or sequence labeling. Feed Forward Neural Networks are capable of achieving good state-of-the-art results in common NLP tasks and have the potential to be much faster than RNN architectures because they require less computations [1]. However, it is clear, that for the most part, RNN have higher accuracy rates than FFNN for NLP.

References

[1] Jan A. Botha, Emily Pitler, Ji Ma, Anton Bakalov, Alex Salcianu, David Weiss, Ryan McDonald, and Slav Petrov. Natural language processing with small feed-forward networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2879–2885, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[2] Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. Long short-term memory neural networks for Chinese word segmentation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1197–1206, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[3] Cícero dos Santos and Maíra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[4] Yoav Goldberg. A primer on neural network models for natural language processing.

[5] Keishen Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding.